KU LEUVEN

FACULTY OF ECONOMICS AND BUSINESS

A branch-and-bound algorithm for shift scheduling with nonstationary demand

Defraeye M, Van Nieuwenhuyse I.

KBI_1322

A branch-and-bound algorithm for shift scheduling with nonstationary demand

Mieke Defraeye* and Inneke Van Nieuwenhuyse
Research Center for Operations Management
Department of Decision Sciences and Information Management
KU Leuven, Belgium
mieke.defraeye@kuleuven.be
inneke.vannieuwenhuyse@kuleuven.be

December 16, 2013

Abstract

Many shift scheduling algorithms presume that the staffing levels, required to ensure a target customer service, are known in advance. Determining these staffing requirements is often not straightforward, particularly in systems where the arrival rate fluctuates over the day. We present a branch-and-bound approach to estimate optimal shift schedules in systems with nonstationary demand and (stochastic) service level constraints. The algorithm is intended for personnel planning in small-scale service systems with limited opening hours (such as small-scale call centers, banks, and retail stores). Our computational experiments show that the algorithm efficiently explores the solution space and quickly finds an optimum (even if an inferior starting solution is used).

Keywords: Time-varying arrival process, Staffing and scheduling, Personnel planning, Capacity analysis, Optimization

1 Introduction

Many shift scheduling algorithms presume that the staffing levels, required to ensure a target customer service, are known in advance: the shift scheduling step then boils down to fitting the min cost shift schedule to the requirements. Determining these staffing requirements, however, is nontrivial at

*Corresponding author.

best, particularly in systems with nonstationary arrival rates. Moreover, this "two-step" approach may result in a suboptimal schedule (Ingolfsson et al., 2010).

This article presents an *integrated* approach to the shift scheduling problem with nonstationary (i.e., time-varying stochastic) demand: different staffing combinations are explored using implicit enumeration, which allows to efficiently estimate the minimum cost shift schedule subject to a service level constraint (the probability that the customer waiting time violates a critical level should not exceed a user-defined target). The algorithm is flexible in the sense that it does not rely on any specific methodology to evaluate the customer service implied by a given shift schedule. We opted to use simulation in our experiments, because (1) it requires virtually no restrictions on the assumptions regarding arrival and service process, (2) it allows us to include real-life complexities of which the impact on customer service cannot easily be estimated analytically, such as customer impatience (abandonments) and the exhaustive service policy (which implies that servers work overtime to finish the customer in service at the time their shift ends), and (3) it allows us to tune the accuracy by changing the number of replications in the simulation model.

The algorithm specifically targets service systems with limited opening hours (so-called *terminating* systems, see Law and Kelton, 2000), and is especially suited for systems with a limited number of operators (such as banks, retail stores, or small call centers). It contributes to the existing literature by proposing straightforward, easy-to-implement rules to efficiently explore the solution space (as opposed to the more complex and time-consuming approach of Atlason et al., 2004, 2008).

Section 2 gives a brief discussion of the related literature. Section 3 presents the formal problem statement. A detailed description of the algorithm is provided in Section 4. Section 5 discusses the computational experiment, and concluding remarks are provided in Section 6.

2 Related literature

Shift scheduling for systems with nonstationary arrival rates has received relatively limited attention in the academic literature. The two-step approach, which fits minimum cost shift schedules to predefined staffing requirements, is by far the most common (see Thompson, 1993, 1997; Sinreich and Jabali, 2007; Izady and Worthington, 2012, among others). The main problem, however, is that the staffing levels required to ensure a target cus-

tomer service level are not straightforward to determine. Moreover, the two-step approach may result in suboptimal shift schedules (Ingolfsson et al., 2002; Henderson and Mason, 1998, 1999b), because several staffing solutions might exist that lead to shift schedules with substantially varying costs. Alternatively, shift scheduling can be done directly based on the time-varying arrival rates (Ingolfsson et al., 2002; Koole and van der Sluis, 2003; Castillo et al., 2009; Helber and Henken, 2010). These approaches avoid the suboptimality that arises by decomposing the problem into two steps. Yet, including quality of service constraints in the shift optimization is not straightforward, hence authors commonly resort to simplifying assumptions (e.g., exponential service and abandonment times).

Our research is closely related to the work of Ingolfsson et al. (2002, 2010) and Atlason et al. (2004, 2008). These articles suggest algorithms to determine low-cost shift schedules with a service level constraint on customer waiting time. Ingolfsson et al. (2002) evaluate schedule performance by numerical integration of the forward differential equations for $M_t/M/s_t$ queues and apply a genetic algorithm to search for good schedules. Ingolfsson et al. (2010) apply a heuristic cutting-plane algorithm and use the randomization method for evaluating schedule performance (Grassmann, 1977; Ingolfsson, 2005; Ingolfsson et al., 2007), which is computationally less expensive but yields similar accuracy (Ingolfsson et al., 2007). Atlason et al. (2004, 2008) suggest a cutting plane method that uses simulation to evaluate customer service, and add cuts based on the estimated (pseudo)gradients of the service level function. This requires substantial computational effort. Atlason et al. (2008) show that their algorithm converges towards an optimal solution as the number of replications grows large; in contrast, both Ingolfsson et al. (2002) and Ingolfsson et al. (2010) are heuristic approaches, that do not guarantee an optimal solution.

The approach developed in this article is easier to implement than the one proposed in Atlason et al. (2004, 2008). Though our approach cannot strictly guarantee the optimum in the exhaustive setting, it will converge to the optimal solution in systems with a preemptive service policy (where service can be interrupted and the customer in service rejoins the queue), as the number of replications grows to infinity.

3 Problem statement and notations

We focus on a single-stage multiserver $M_t/G/s_t + G$ queue, as depicted in Figure 1. The current time is represented by t and ranges between 0 and

time horizon T (i.e., the opening hours of the service system). Customer arrivals have a time-varying arrival rate λ_t (in our numerical experiments, we assume Poisson arrivals, though this choice is by no means restrictive). The service process is generally distributed with per server service rate μ; the abandonment process is generally distributed with rate θ.

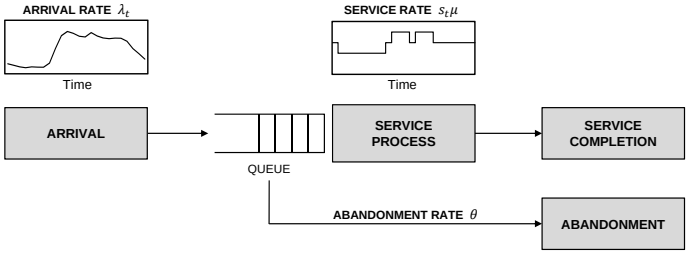

Figure 1: Schematic representation of a single-stage queueing system with time-varying demand

The main objective is to estimate an optimal shift schedule, such that the target customer service is achieved at minimum cost. The cost is measured in man-hours. In line with the related literature (Feldman et al., 2008; Ingolfsson et al., 2010; Campello and Ingolfsson, 2011; Izady and Worthington, 2012), customer service is measured by the *virtual waiting time* W_t at given time instants t, i.e., the waiting time that an infinitely patient (fictive) customer encounters upon arrival at time t (Gross et al., 2008; Le Minh, 1978; Mandelbaum and Momğilović, 2008). More formally, let $\mathbf{t}_p = \{0, \Delta_p, 2\Delta_p, \ldots, T - \Delta_p\}$ represent the set of time instants at which performance is evaluated (the notations are illustrated in Figure 2). We then require the following hard constraint to be met:

$$\Pr(W_t > \tau) \leq \alpha \quad \text{for all } t \in \mathbf{t}_p\,, \tag{1}$$

with τ the maximum allowed waiting time, and α the target probability of excessive waiting. The validity of this constraint is checked by simulation. Note that for $\tau = 0$, Expression (1) corresponds to the delay probability.

Capacity changes can only take place at specific points in time, i.e., at the start of a *staffing interval*. Staffing intervals have length Δ_s. The set of staffing interval indices is $\mathbf{I}_s = \{1, \ldots, I_s\}$ with $I_s \equiv T/\Delta_s$ (see Figure 2). $\mathbf{t}_s = \{0, \Delta_s, 2\Delta_s, \ldots, T - \Delta_s\}$ contains the staffing interval start times, for all $i_s \in \mathbf{I}_s$ (with $\mathbf{t}_s \subseteq \mathbf{t}_p$).

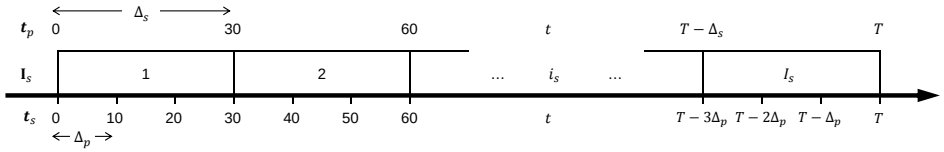

Figure 2: Illustration of notation for staffing intervals (length Δ_s) and performance intervals (length Δ_p)

Let vector $\mathbf{s} = \{s_1, ..., s_{I_s}\}$ represent the staffing vector, containing the number of operators in each staffing interval. Assume that W different predefined shift types exist. For any staffing vector $\mathbf{s}$, the min-cost shift solution can be determined by solving the following basic set covering problem (as introduced in Dantzig (1954)):

$$\min \quad \sum_{j=1}^{W} c_j w_j \tag{2}$$

$$\text{s.t.} \quad \sum_{j=1}^{W} a_{j,i_s} w_j \geq s_{i_s} \qquad \forall i_s \in \mathbf{I}_s \tag{3}$$

$$w_j \geq 0 \text{ and integer} \quad \forall j = 1, \ldots, W \tag{4}$$

The objective function denotes the total shift cost, with c_j the cost of shift j (expressed in man-hours). In constraint (3), the indicator a_{j,i_s} equals 1 if interval i_s is an active period in shift j and equals 0 otherwise. Constraint (4) imposes non-negativity on the shift solution vector $\mathbf{w} = \{w_1, ..., w_W\}$, that defines how many workers are assigned to each shift type. The actual number of operators implied by a given shift vector $\mathbf{w}$ is expressed as $\mathbf{s}_w = \{s_{w,1}, \ldots, s_{w,I_s}\}$. Note that different $\mathbf{w}$ may give rise to the same $\mathbf{s}_w$, and that $\mathbf{s}_w$ will tend to differ from $\mathbf{s}$, as the shift schedule will tend to introduce slack on the first constraint in Problem (2-4).

The overall objective is to minimize the shift cost c_w, while ensuring that the related shift vector w satisfies the performance constraint in Expression (1). The abandonment cost is not included in the objective, but instead is influenced implicitly through the performance constraint: as abandonment behavior will increase as the waiting times grow, τ should be small compared to $1/\theta$ if abandonments are to be avoided.

The exhaustive service policy implies that servers will work overtime at the time their shift ends, to finish the ongoing service instance (if any). As such, customers cannot be transferred between servers. Note that this does

not completely match the exhaustive service policy applied in Atlason et al. (2008), which only allows for overtime when the overall scheduled capacity decreases (i.e., when the servers that go off duty are not replaced by new servers).

4 Branch-and-bound algorithm

In this section, we develop a branch-and-bound algorithm for shift scheduling with nonstationary arrival rates. Section 4.1 discusses how the search tree is constructed. Section 4.2 describes in detail how this tree is explored, and which rules are applied to guide the search procedure. The algorithm's pseudocode is given in Appendix A.

4.1 Tree structure

The construction of the tree requires the following three staffing vectors as input:

- an initial feasible solution $\mathbf{s}^{\mathsf{init}}$: any staffing vector that satisfies the performance constraint qualifies as initial feasible solution. A tighter initial feasible solution, however, speeds up convergence. The corresponding min-cost shift vector, $\mathbf{w}^{\mathsf{init}}$ (with cost c_w^{init}), is obtained as the integer programming solution to Problem (2-4).

- a lower bound vector $\mathbf{s}^{\mathsf{LB}}$: this vector contains the lower bound on the staffing requirements for each interval $i_s \in \mathbf{I}_s$. Any staffing vector with capacity smaller than $\mathbf{s}^{\mathsf{LB}}$ in at least 1 interval, can never be feasible. Appendix B details how to obtain $\mathbf{s}^{\mathsf{LB}}$.

- an upper bound vector $\mathbf{s}^{\mathsf{UB}}$: All solutions for which $s_{i_s} > s_{i_s}^{\mathsf{UB}}$ in at least one staffing interval yield a staffing cost that exceeds c_w^{init}, and should not be considered. Appendix B details how to obtain $\mathbf{s}^{\mathsf{UB}}$.

An illustration of the tree structure is presented in Figure 3, for $I_s = 3$. Each node in the tree represents a staffing vector $\mathbf{s}$, with corresponding staffing cost c_s. The root node of the tree is initialized to $\mathbf{s}^{\mathsf{LB}}$ (as staffing vectors with capacity smaller than $\mathbf{s}^{\mathsf{LB}}$ in at least 1 interval are infeasible, they need not be considered in the search tree).

Starting from the root node, $\mathbf{s}$ is increased throughout the search tree. Each level in the tree is denoted by its depth $d = 0, \ldots, I_s$ ($d = 0$ represents the depth of the root node). Child nodes are generated from a parent node

by adding capacity to a given staffing interval (see Figure 4): child nodes at level $d+1$ differ in capacity from the parent node in staffing interval $d+1$, where the staffing level takes values between its lower bound, s_{d+1}^{LB} and its upper bound, s_{d+1}^{UB}. The staffing levels in the other intervals are identical to those of the parent node.

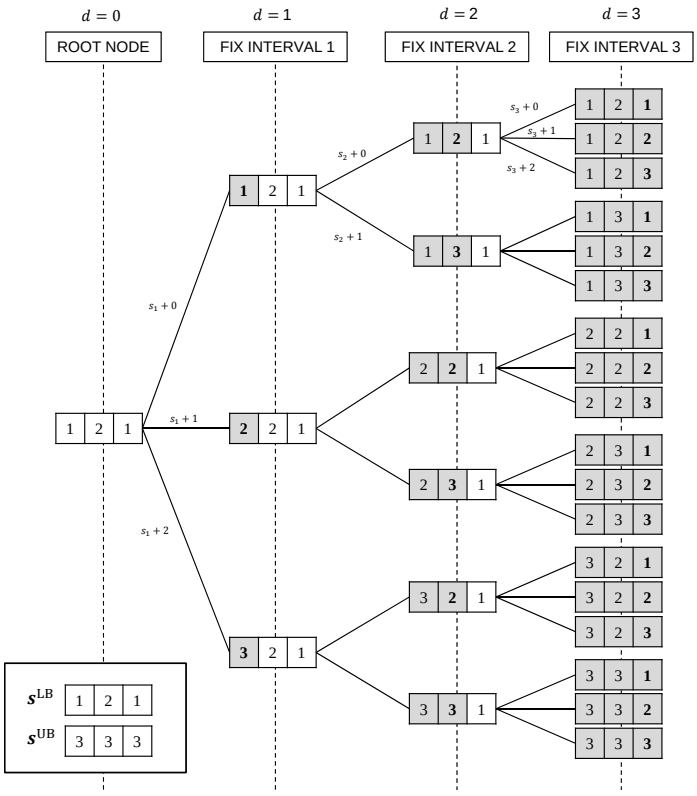

Figure 3: Example tree structure ($I_s = 3$)

The total number of nodes in the tree (denoted by $\mathcal{S}$) is equal to:

$$\mathcal{S} = \prod_{i_s=1}^{I_s} \left(s_{i_s}^{\mathsf{UB}} - s_{i_s}^{\mathsf{LB}} + 1 \right). \tag{5}$$

As explained in Section 4.2, defining the search tree in terms of staffing vectors enables an efficient exploration of the solution space. For each staffing vector $\mathbf{s}$, the corresponding min-cost shift solution $\mathbf{w}$ can be retrieved by solving Problem (2-4). Note that the $\mathbf{s}$ vectors themselves are not

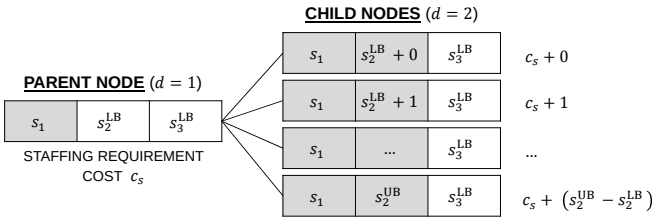

Figure 4: Illustration: Branching to a lower level ($I_s = 3$)

checked for feasibility with respect to the performance constraint: only the feasibility of $\mathbf{w}$ is relevant. By implicitly enumerating all staffing vectors, the algorithm avoids the suboptimality that is inherent in the traditional two-step approach.

4.2 Node exploration

For any given parent node, child nodes are considered in increasing order of c_s (i.e., from top to bottom, in Figure 3). The tree is explored in a depth-first manner: after checking a node at depth d, the algorithm branches to the lowest cost child node at levels $d+1, d+2, \ldots$ etc. If the lowest level is reached ($d = I_s$) and all child nodes of the current parent node have been explored, we *backtrack*: the algorithm then returns to the previous level and continues with the next unexplored node in the tree. Note that in Figure 3, the top child node at level $d+1$ duplicates the parent node at level d; these duplicates are shown for completeness and are not explored.

To limit the number of nodes for which we need to effectively simulate the customer service level, we implement rules to *fathom* nodes. A node is fathomed if it is discarded from the search procedure, along with all its underlying child nodes. Throughout the algorithm, the best (feasible) shift vector found so far is stored ($\mathbf{w}^*$, with shift cost c_w^*). At the start of the algorithm, $\mathbf{w}^*$ is initialized to $\mathbf{w}^{\text{init}}$.

Every node in the tree is evaluated according to the rules summarized in Figure 5. Sections 4.2.1, 4.2.2 and 4.2.3 describe the computationally inexpensive rules (Fathom$[c_s]$, Fathom$[c_w]$, Fathom$[c_w^{\mathcal{R}}]$, and Fathom$[i_s^e]$) used in steps 1-3 to identify nodes that can be fathomed. Only shift vectors that cannot be fathomed in these steps, are simulated in step 4 (see Section 4.2.4). Based on the simulation outcome, two additional fathoming rules (Fathom$[\mathbf{w}^*]$ and Fathom$[i_s^e]$) are applied to further constrain the solution space.

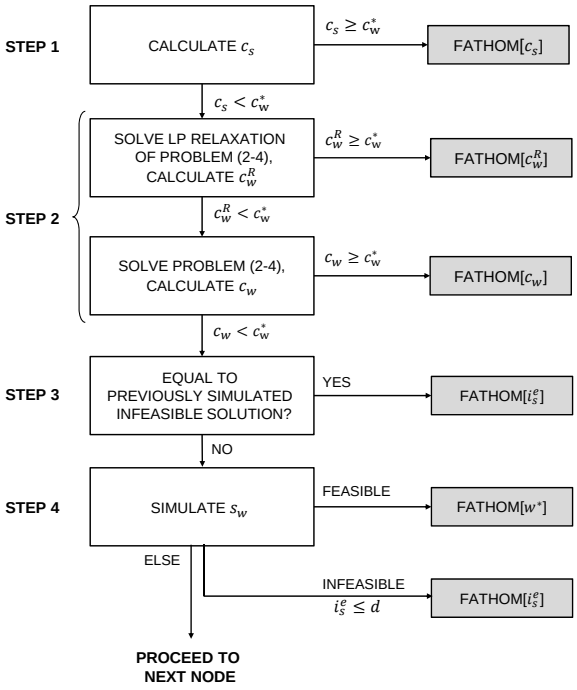

Figure 5: Node exploration

4.2.1 Step 1: Evaluate staffing cost c_s

For any node **s**, we first evaluate its staffing cost c_s: if $c_s \geq c_w^*$ (with c_w^* the best *shift* cost so far), then node **s** can be fathomed along with its underlying nodes and all unexplored child nodes from the same parent node. Indeed, all child nodes of **s** have a staffing cost which is at least as large as c_s (as illustrated in Figure 4), so their corresponding shift cost cannot be smaller than c_w^*. As nodes at a given level are explored in increasing order of c_s, the same is valid for the remaining unexplored child nodes with the same parent node as **s**. The algorithm then proceeds to the next unexplored node in the tree: this can be a node at depth $d-1$ along the same branch as the parent node, or a node higher in the tree (if backtracking takes place).

This rule is referred to as Fathom[c_s]. Due to its low computational effort, it is used as a first criterion to eliminate parts of the solution space that cannot contain an optimum.

4.2.2 Step 2: Evaluate shift cost c_w

If $\mathbf{s}$ could not be fathomed in step 1, the minimum shift cost c_w is determined. We first solve the LP relaxation of Problem (2-4); let's denote its shift cost by $c_w^{\mathcal{R}}$. If $c_w^{\mathcal{R}} \geq c_w^*$, then node $\mathbf{s}$ is fathomed along with its underlying nodes, and all unexplored child nodes from the same parent node (the argument is analogous to the one presented in step 1). As in step 1, the algorithm proceeds to the next unexplored node in the tree. Only when $c_w^{\mathcal{R}} < c_w^*$, Problem (2-4) is solved with the integrality constraints included; when $c_w \geq c_w^*$, again node $\mathbf{s}$ is fathomed along with its underlying nodes, and all unexplored child nodes from the same parent node. These fathoming rules are referred to as Fathom$[c_w^{\mathcal{R}}]$ and Fathom$[c_w]$.

A limitation of our model is that it selects only one min-cost shift vector in each node, as such, possible alternative optima to Problem (2-4) are not accounted for. In systems with an exhaustive service policy, the start and end times of shifts impact the performance estimates. Alternative shift vectors with identical cost may result in slightly different performance estimates in such a setting (even if the capacity profile $\mathbf{s}_w$ is identical over the day), which could cause the algorithm to miss the optimum. This limitation especially holds for highly utilized systems with long service times, because the exhaustive service policy is most prominent in such settings.

4.2.3 Step 3: Check if w was simulated before

Different $\mathbf{s}$ vectors can result in identical $\mathbf{w}$ vectors. As such, it is plausible that a given $\mathbf{w}$ vector with $c_w < c_w^*$ has already been simulated at a previous node. As simulations can be computationally expensive, we store each previously simulated infeasible $\mathbf{w}$ vector in a set (denoted by $\mathbf{B}$), along with information on the first time instant at which the performance constraint was violated:

$$t^e = \min\{t \in \mathbf{t}_p : \Pr(W_t > \tau) > \alpha\}. \tag{6}$$

If $\mathbf{w} \in \mathbf{B}$ for a given staffing vector $\mathbf{s}$, the t^e value allows to detect other infeasible staffing vectors, at least in systems with limited opening hours. Indeed, to attain an acceptable waiting time $W_{t^e} \leq \tau$, a customer arriving at time t^e needs to enter service at $t^e + \tau$ at the latest. If we let i_s^e denote the staffing interval that contains $t^e + \tau$, all $\mathbf{s}'$ for which $s'_{i_s} \leq s_{i_s}$ for all $i_s \in \{1, \ldots, i_s^e\}$ are infeasible as well, irrespective of the capacity in intervals $i_s > i_s^e$ (note that this does not necessarily hold in nonterminating systems). This observation is used to define a last fathoming rule, termed Fathom$[i_s^e]$. Consider an infeasible node at level d, with corresponding i_s^e. Three cases

can then be distinguished; these are illustrated in the example provided in Figure 6 (for $I_s = 5$):

1. $i_s^e < d$ (see Figure 6a). In that case, the parent node at depth i_s^e can be fathomed and the algorithm proceeds with the next unexplored node at level i_s^e.
2. $i_s^e = d$ (see Figure 6b). In that case, the node at depth d can be fathomed and the algorithm proceeds with the next unexplored node at level d.
3. $i_s^e > d$ (see Figures 6c and 6d). In that case, we branch to the next unexplored child node at level i_s^e.

As a result, the algorithm each time augments the capacity in the interval that causes the performance constraint to be violated, i_s^e. Evidently, any violation might also be solved by increasing capacity in prior intervals $i_s < i_s^e$. These solutions will be encountered later in the algorithm, when the algorithm backtracks to levels $d < i_s^e$.

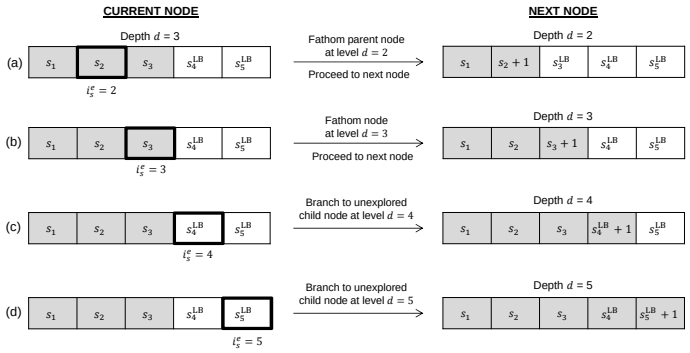

Figure 6: Example fathoming and branching based on infeasibility (5 staffing intervals, $d = 3$)

Note that the Fathom$[i_s^e]$ rule is particularly straightforward to apply given that the search tree is defined in terms of **s** vectors.

4.2.4 Step 4: Evaluate shift vector $\mathbf{s}_w$ through simulation

For nodes that could not be fathomed based on one of the rules in steps 1-3, the feasibility of the min cost shift vector **w** with respect to the service level constraint in (1) is evaluated by means of simulation (see Appendix C for further details on the evaluation procedure).

If **w** is feasible, a new optimum has been found (as c_w is smaller than c_w^*). Again, all unexplored child nodes that share the same parent can be fathomed as these cannot improve the optimum. This fathoming rule is referred to as Fathom[$\mathbf{w}^*$]. As in steps 1 and 2, the algorithm proceeds with the next unexplored node in the search tree.

If, on the contrary, **w** is infeasible, the Fathom[i_s^e] rule is applied (see Section 4.2.3), and the vector **w** is added to set **B**.

5 Results

The approach described in Section 4 is tested on a set of 972 problem instances. All experiments are performed on an Intel I7 3.40 GHz computer, with 8 GB RAM. The experimental setup is described in Section 5.1. Section 5.2 discusses the algorithm's computational performance with respect to the number of nodes explored, and the improvement in the shift cost obtained with respect to the initial solution.

5.1 Experimental setting

Table 6 contains the parameter settings of the test set. We assume that the service system is open 12 hours per day and that the arrival rate follows a sinusoidal pattern with 2 peaks per day, fluctuating around the average rate $\bar{\lambda}$:

$$\lambda_t = \bar{\lambda}\left(1 + RA \sin\left(\frac{2\pi t}{8}\right)\right)$$

where RA denotes the relative amplitude of the arrival rate, and with t expressed in hours.
The service and abandonment distributions are assumed to be of the same type in each of the test instances: either both are exponential ($SCV = 1$), 2-phase Erlang ($SCV = 0.5$), or 2-phase Coxian ($SCV = 2$).

The shift sets are provided in Appendix D. Each shift is 4, 6, or 8 hours long and may include a one-hour break. This yields a set of 5 shifts for $\Delta_s = 240$ min, a set of 12 shifts for $\Delta_s = 120$ min, and a set of 45 shifts for $\Delta_s = 60$ min (the latter is identical to the shift set of Ingolfsson et al., 2010). The algorithm is terminated if an estimated optimal solution has not been found after 25,000 nodes have been simulated in the tree exploration phase.

Parameter	**Parameter values**
Service rate μ (customers/hour)	$\{1, 2, 4\}$
Offered load $\bar{\lambda}/\mu$	$\{5, 10, 15\}$
Relative amplitude arrival rate RA	$\{0.5, 1\}$
Abandonment rate θ (customers/hour)	$\{0, \mu\}$
Max wait τ (min)	$\{0, 10, 20\}$
Squared coefficient service and abandonment times (SCV)	$\{0.5, 1, 2\}$
Staffing interval Δ_s (min)	$\{240, 120, 60\}$
Performance interval Δ_p (min)	5
Number replications per simulation R	2500
Target α	0.2

Table 1: Experimental setting

5.2 Algorithm performance

In our computational experiments, we select $\mathbf{s}^{\text{init}}$ by means of the ISA(τ) algorithm (Defraeye and Van Nieuwenhuyse, 2013), a staffing heuristic which ensures a fairly tight and feasible staffing solution[1]. Table 2 contains statistics on the number of simulation runs needed to find the initial feasible solution and lower bound ("preprocessing" phase). It reveals that the initial feasible solution and the lower bound can be derived with a very small number of simulations.

	Min	Median	Max	Average
Simulations for initial feasible solution	5	16	36	16
Simulations for lower bound	6	18	117	26

Table 2: Statistics on the preprocessing phase, over all test instances

Figures 7(a) and (b) confirm these findings; they show the number of nodes explored with low computational effort (steps 1 to 3 in Figure 5) and high computational effort (step 4 in Figure 5), for each problem instance that could be solved to optimality. Figure 7(a), that presents the number of nodes explored as a percentage of the total solution space (given by Expression 4.1). It shows that the algorithm is efficient: only a minor percentage of the nodes in the solution space are explored during the algorithm. Figure 7(b) depicts

[1]We use a slightly different stop criterion than Defraeye and Van Nieuwenhuyse (2013): the algorithm stops when the solution was already evaluated before, or when $\max_{t \in \mathbf{t}_p}\{\Pr(W_t > \tau)\}$ deviates less than 5% from its average value over the past 25 iterations.

the absolute numbers, showing more explicitly that the number of nodes requiring simulation is only a fraction of the nodes that are explored with low computational effort (i.e., most observations lie below the diagonal).

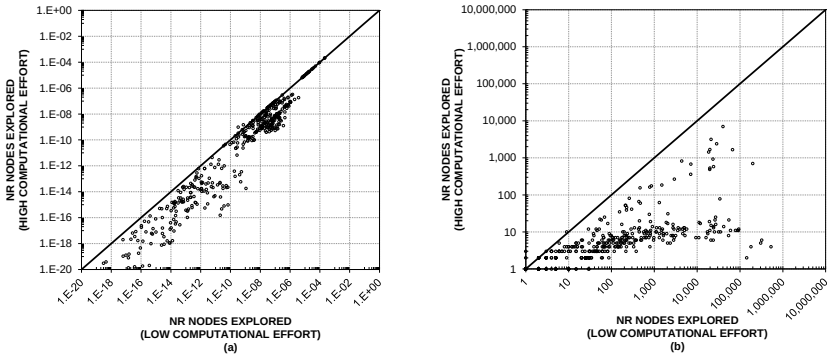

Figure 7: Number of explored nodes fathomed by low and high effort fathoming rules, (a) as a percentage of the total solution space, (b) in absolute values.

Figure 8 analyzes the computational effort in the tree exploration phase (measured by the number of nodes requiring simulation) versus the improvement in cost obtained with respect to the initial feasible solution. This cost improvement is determined by:

$$\text{Cost improvement} = \frac{(c_w^{\text{init}} - c_w^*)}{c_w^{\text{init}}}. \tag{7}$$

The top row of Figure 8 shows that in 15.74% of test instances, the algorithm terminates after 25,000 simulation runs (so, without a guarantee that there is no better solution to be found). It appears that the size of the solution space is a decisive factor here. All these instances allowed for 12 staffing intervals ($\Delta_s = 60$). Moreover, as detailed in Table 3, the performance decreased as the relative amplitude of the arrival rate and/or the offered load increased (which implies that higher staffing will likely be needed to satisfy the customer service constraint).

The remaining 84.26% of instances were solved to optimality, as summarized in the bottom matrix of Figure 8. Table 4 gives further details on these instances, analyzing the performance of the algorithm across different parameter settings. We compare only instances that were solved to optimality for each value of a particular parameter (all else equal); the last column in the table contains the number of instances.

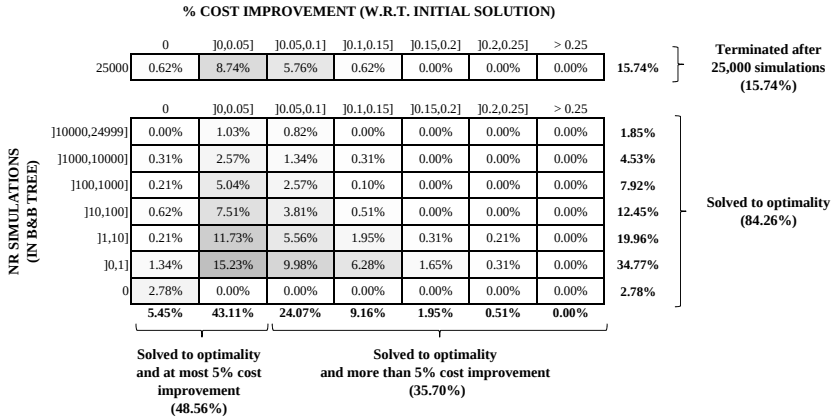

Figure 8: Simulation runs performed in branch-and-bound tree vs. percent improvement over the initial solution

	Offered load		
	5	10	15
RA = 0.5	96%	59%	35%
RA = 1	78%	30%	19%

Table 3: Percentage of instances solved to optimality, for each combination of relative amplitude and offered load (for $\Delta_s = 60$)

As indicated by the first column of Figure 8, the initial solution turns out to be optimal in 5.45% of the instances (0% cost reduction); verifying this may require a considerable number of simulations though. As shown in Table 4, the probability that the initial solution is optimal increases as (1) the service rate increases, (2) the offered load decreases, (3) the abandonment rate increases, (4) the waiting time target is less stringent, (5) the service and abandonment processes are less variable, and (6) the staffing intervals are large. This is not surprising, as all these factors limit the solution space (both the number of staffing intervals and the capacity required), so it can be expected that the heuristic solution is more likely to coincide with the estimated optimum.

Overall, a majority of instances (approx. 70%) could be solved to optimality quickly (less than 100 simulation runs, see Figure 8). The improvement with respect to the initial solution turned out to be relatively small

in most cases (improvements of more than 5% were only observed in about 35.70% of the instances, with improvements exceeding 15% in only 2.47% of the instances). This mainly confirms that the quality of the initial solution, as generated by the ISA(τ) algorithm (Defraeye and Van Nieuwenhuyse, 2013), is high (the related c_w^{init} tends to be close to the optimal shift cost). As confirmed by Table 4, cost improvements exceeding 5% were especially likely in settings with low service rates, or high variability in the service and abandonment processes.

Table 4 also provides more general insights into the optimal shift cost, across the different parameter settings. As expected, c_w^* increases as the offered load increases, and the relative amplitude of the arrival process increases. This is intuitive, as both factors imply that more capacity will be needed to meet the customer service constraint. Abandonments, by contrast, reduce the load on the system, and thus have a beneficial impact on the optimal shift cost. Furthermore, the staffing interval length plays a role: short staffing intervals provide more flexibility to the shift schedule, which tends to lead to lower optimal costs.

Finally, as observed before, the computational effort (as measured by the number of simulations performed) is highly sensitive to the size of the solution space, with the staffing interval length having a particularly large impact.

5.3 Impact number of replications

Any inaccuracies in the estimated customer service may affect the solution that is returned by the algorithm. In particular in steps 3 and 4 of Figure 5, nodes are fathomed based on the service estimates, so inaccurate estimates may cause the algorithm to settle at a wrong optimum. As we use simulation to evaluate customer service, the estimation accuracy is impacted by the number of replications R. In this section, we compare $R = 100$ versus $R = 2500$, and assess the extent to which the difference in accuracy affects the computational effort required to run the algorithm to completion, and the observed cost difference at the final solution, for those instances that were solved to optimality.

More specifically, the difference in computational effort is measured through the number of simulation runs required (during the preprocessing and tree exploration stages):

$$\boldsymbol{\Delta}\text{SIM} \equiv \text{SIM}(R = 100) - \text{SIM}(R = 2500). \tag{8}$$

Parameter	Values	Average optimal cost c_w^*	Average number of simulations (tree exploration)	Average % of instances where initial solution is optimal	Average % of instances with cost reduction $> 5\%$	Number of instances
Service rate (μ)	1	162.333	554.462	0.000	0.831	249
	2	166.064	315.137	0.016	0.353	249
	4	159.108	72.281	0.161	0.169	249
Offered load (λ/μ)	5	96.559	33.751	0.110	0.486	245
	10	168.931	157.376	0.065	0.416	245
	15	239.078	455.482	0.033	0.327	245
Relative amplitude (RA)	0.5	121.040	144.007	0.073	0.442	392
	1	137.248	728.343	0.091	0.464	392
Abandonment rate (θ)	0	170.119	550.000	0.026	0.537	390
	μ	151.343	331.750	0.112	0.269	390
Maximal acceptable wait (τ)	0	198.754	124.873	0.056	0.349	252
	10	152.111	141.131	0.063	0.397	252
	20	134.147	566.071	0.091	0.548	252
SCV of service and abandonment process	0.5	169.629	212.367	0.110	0.295	264
	1	162.663	372.201	0.061	0.383	264
	2	153.902	400.530	0.030	0.595	264
Staffing interval length (Δ_s)	60	128.766	2715.509	0.070	0.287	171
	120	141.029	34.333	0.099	0.298	171
	240	143.766	22.094	0.135	0.374	171

Table 4: Sensitivity to system parameters ($R = 2500$), for instances solved to optimality

The cost difference (in percent) is determined as:

$$\mathbf{\Delta} c_w^* \equiv \frac{c_w^*(R = 100) - c_w^*(R = 2500)}{c_w^*(R = 2500)}. \tag{9}$$

Figure 9 shows that the difference in computational effort varies widely (with 5% and 95% percentiles equal to -662.4 and 251.4 respectively). The difference in cost, by contrast, is far less outspoken: increasing the number of replications has only a limited impact on the cost of the final solution (see Figure 10). Using $R = 100$ yields a c_w^* that is 1.84% higher on average (the 5% and 95% percentiles equal -1.69% and 7.14% respectively).

The time-average of the confidence interval halfwidth is about 4% on average (for $R = 100$) and 0.8% (for $R = 2500$). We found that in 32.6% of the instances, the final solutions obtained with $R = 100$ appear to be infeasible if they are evaluated with $R = 2500$. Though the performance constraint was typically violated in only a limited number of performance intervals, this shows that R should be large in settings where the performance constraint is strict.

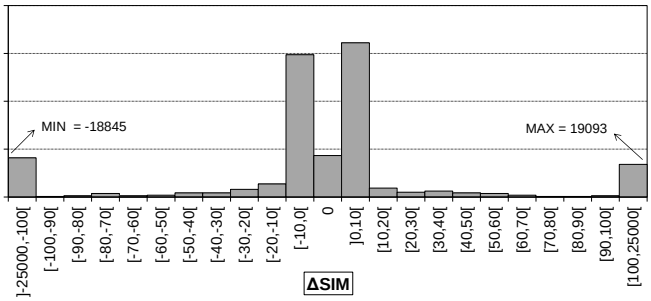

Figure 9: Sensitivity of the optimal solution to number of replications: difference in number of simulation runs

5.4 Impact of the initial solution

In all computational results shown so far, the initial solution was generated by the ISA(τ) algorithm (Defraeye and Van Nieuwenhuyse, 2013). As evident from the results, this initial solution tends to be of high quality. In this section, we explore how a lower-quality initial solution affects the number of simulations required to terminate the algorithm, and the speed with which an estimated optimal solution is found. Because the algorithm is stopped

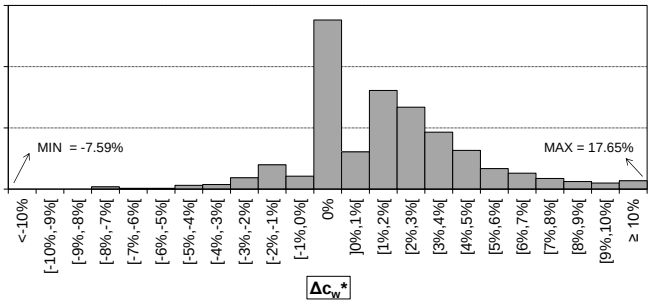

Figure 10: Sensitivity of the optimal solution to number of replications: difference in optimal shift cost

after a fixed number of simulated nodes, it is important that good feasible solutions are found quickly, even if their optimality is not guaranteed. Ideally, the algorithm's speed in finding the optimal solution should not be impacted too severely by the quality of the initial solution.

The purpose of the initial feasible solution is twofold: it enables using the fathoming rules defined in Section 4.2 (it provides a value for c_w^{init}), and speeds up the search for the lower bound on the staffing requirements (which defines the root node of the tree). In this section, we apply an alternative initial solution that is simpler to calculate (it requires no simulation runs) but results in a higher initial shift cost. More specifically, $\mathbf{s}^{\mathsf{init}}$ is obtained as the smallest staffing vector that satisfies the delay probability constraint (i.e., τ equal to 0) in a stationary $M/M/s$ model with arrival rate $\lambda_{\mathsf{max}} = \max\{\lambda_t : t \in [0,T]\}$. This vector is feasible in the corresponding $M_t/M/s_t + M$ model (although it is probably very costly). In our experiments, the feasibility remains valid for general service and abandonment times, due to the large amount of excess capacity that is added due to the overly restrictive assumptions that are used (i.e., no abandonments, the use of λ_{max} and τ equal to 0).

Figure 11 contains the difference in the total number of simulation runs in the algorithm, for the instances that were solved to optimality (the alternative initial solution is indicated by $M/M/s$). The figure reveals that the total of simulations tends to be lower for ISA(τ). A paired t-test showed that the difference is significant (with $p < 0.01$). As such, the simulation runs required to determine the ISA(τ) solution result in a more than proportional reduction of the number of simulations needed to run the algorithm to completion. The algorithm succeeds in finding good solutions quickly, irrespective of the start solution: The differences in total simulation effort

are generally small, even though the initial staffing cost c_s^{init} corresponding to the $M/M/s$-based solution may be substantially higher. Indeed, Table 5 shows that the best solution is typically found after a low number of simulation runs for both initial solutions (though the algorithm may require a substantial number of simulation runs to terminate).

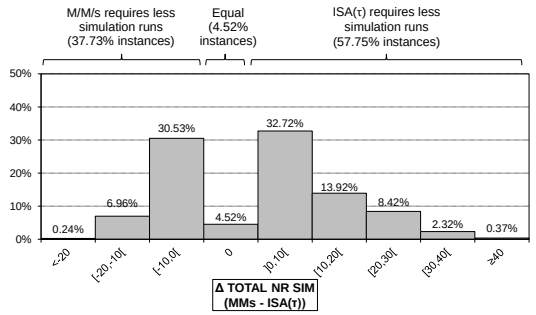

Figure 11: Difference in total number of simulations ($M/M/s$ - ISA(τ))

	Min	5% percentile	Median	95% percentile	Max	Average
ISA(τ)	11	18	35	2977	22671	606.04
$M/M/s$	16	18	39	2997	22686	609.68

Table 5: Comparison: total number of simulations required to reach an estimated optimum.

Figure 12 provides further details on how the difference in initial staffing cost affects the algorithm's performance. It shows the percentage of test instances (solved to optimality), as a function of the difference in initial staffing cost and the total number of simulations. The figure reveals that the $M/M/s$-based solution outperforms the ISA(τ) solution only if its staffing cost is close to that of the ISA(τ) solution (note that this information is not available in advance). In that case, determining the ISA(τ) solution is not worthwhile the additional simulation effort. The staffing costs, however, often differ greatly: in 17% of instances, $c_s^{\text{init}}(M/M/s)$ is more than twice as large as $c_s^{\text{init}}(\text{ISA}(\tau))$. In those settings, the ISA(τ) solution clearly outperforms the $M/M/s$-based solution.

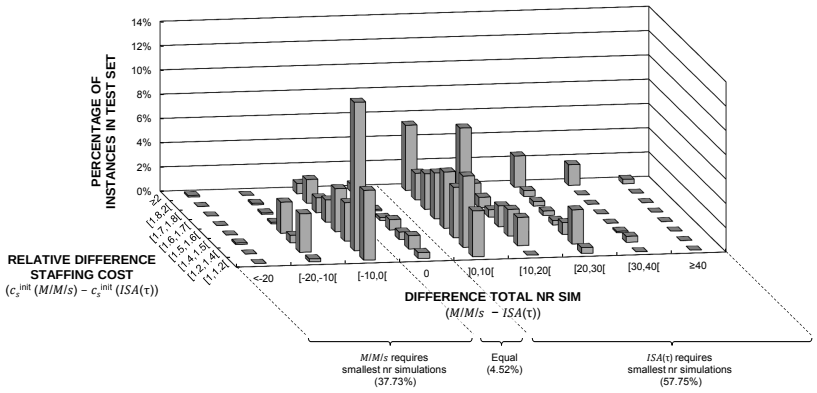

Figure 12: Impact of the initial solution: classification of test instances based on difference in number of simulations and difference in initial staffing cost

6 Concluding remarks and limitations

We present an implicit enumeration approach to estimate optimal shift schedules in terminating systems with nonstationary arrival rates and service level constraints. The results show that the algorithm is efficient in exploring the solution space, though the computational effort increases significantly as the number of staffing intervals and the server requirements per interval increase. Consequently, the algorithm is best suited for small-scale systems, with a limited number of operators.

The algorithm is efficient and an estimated optimum is typically found quickly (even if an inferior start solution is used). The algorithm does not depend on a particular methodology to evaluate the service level constraints; in principle, any type of methodology can be used. However, the quality of the optimal solution proposed by the algorithm evidently depends on the *accuracy* of the customer service estimates.

The optimal solution found by our method is an *estimated* optimum because discrete-event simulation is used to estimate the service levels, and because alternative optima for Problem (2-4) are not accounted for. As is discussed in Section 4.2.2, the existence of alternative optima could cause the algorithm to miss the optimum in settings with an exhaustive service policy. This limitation especially holds for highly utilized systems with long service times, because the exhaustive service policy is most prominent in such settings. Though our approach cannot strictly guarantee the optimum in the exhaustive setting, it will converge to the optimal solution in systems

with a preemptive service policy (where service can be interrupted and the customer in service rejoins the queue), as the number of replications grows to infinity.

In future research, we plan to use our method to evaluate the solution quality of heuristic approaches available in the literature (such as Ingolfsson et al., 2002, 2010, among others), and to extend the approach towards nonterminating settings.

Acknowledgments

This research was supported by the Research Foundation-Flanders (FWO) (grant no G.0547.09).

Appendix A

Algorithm 1 Simulation-based branch-and-bound algorithm

```
function ExploreTree()

    Set initial feasible staffing vector s^init
    Determine w^init

    w* ← w^init     // Initialize best feasible solution so far
    s*_w ← s^init_w
    c*_w ← c^init_w

    Set bounds s^LB and s^UB

    s ← s^LB     // Initialize root node
    d ← 0     // Initialize depth

    repeat
        Determine c_s
        if c_s ≥ c*_w then
            Fathom[c_s]:
            s_d ← s^LB_d     // Restore capacity in interval d
            d ← d − 1     // Go to parent node.
            Backtrack()
        else
            Solve Problem (2-4) to obtain {w, c_w, s_w}
            if c_w ≥ c*_w then
                Fathom[c_w]:
                s_d ← s^LB_d     // Restore capacity in interval d
                d ← d − 1     // Go to parent node.
                Backtrack()
            else
                Check if w has been simulated before
                if w ∈ B : then
                    i^e_s ← i^e_s of previously simulated solution
                    if d < i^e_s then
                        Branch(i^e_s)     // Proceed to child node on level i^e_s
                    else   //  d ≥ i^e_s so branching to underlying nodes is useless
                        Fathom[B] :
                        while (d > i^e_s) do
                            s_d ← s^LB_d     // Restore capacity in interval d
                            d ← d − 1     // Go 1 level back
                        end while
                        Backtrack()
                    end if
                else
                    Simulate s_w to obtain i^e_s
                    if Feasible then
                        w* ← w     // Update best solution so far
                        s*_w ← s_w
                        c*_w ← c_w

                        Fathom[w*]:
                        s_d ← s^LB_d     // Restore capacity in interval d
                        d ← d − 1     // Go to parent node.
                        Backtrack()
                    else   // Infeasible solution
                        if d < i^e_s then
                            Branch(i^e_s)     // Proceed to child node on level i^e_s
                        else   //  d ≥ i^e_s so branching to underlying nodes is useless
                            Fathom[i^e_s] :
                            while (d > i^e_s) do
                                s_d ← s^LB_d     // Restore capacity in interval d
                                d ← d − 1     // Go 1 level back
                            end while
                            Backtrack()
                        end if
                    end if
                end if
            end if
        end if
    until (d = 0)
end function
```

Algorithm 2 Branch

```
function BRANCH(d')    // branch to child node on level d'
    d ← d'    // Increment depth
    if s_d < s_d^UB then
        s_d ← s_d + 1    // Augment capacity in interval d
    else
        Backtrack()
    end if
end function
```

Algorithm 3 Backtrack

```
function BACKTRACK()    // Return to previous level and proceed with next node
    if d ≠ 0 then
        while (s_d = s_d^UB and d > 0) do
            s_d ← s_d^LB    // Restore capacity in interval d
            d ← d − 1    // Go 1 level back
        end while
        if d ≠ 0 then
            s_d ← s_d + 1    // Proceed to next node on the same level
        end if
    end if
end function
```

Appendix B: Bounds

Lower bound The LB is determined as follows: in each staffing interval i_s, $s_{i_s}^{\mathsf{LB}}$ is set equal to the smallest capacity level that is needed to meet the performance constraint, assuming that infinite capacity is available in all other staffing intervals. Ingolfsson et al. (2010) suggest a similar approach, but start from an empty system. While Ingolfsson et al. (2010) use bisection search to obtain $s_{i_s}^{\mathsf{LB}}$ for each i_s, we opt to make unit-size decreases starting from the feasible (heuristic) solution $s_{i_s}^{\mathsf{init}}$ obtained through ISA(τ)); as $s_{i_s}^{\mathsf{init}}$ tends to be relatively tight, we found that this approach finds the lower bound with fewer evaluations than bisection search. Note that, for $\tau \geq \Delta_s$, it suffices to set s^{LB} equal to 1 (assuming that at least 1 server should be available at all times): as τ spans multiple staffing intervals, the capacity shortage in any given interval is compensated by the infinite capacity in the following interval.

Upper bound The vector $\mathbf{s}^{\mathsf{UB}}$ contains an upper bound on the staffing requirement in each staffing interval. It is constructed based on the initial shift cost c_w^{init}. For each interval i_s, the cheapest shift that can be active in that interval is selected. Let this shift be represented by j_{min}, with shift

cost $c_a^{j_{\mathsf{min}}}$. The upper bound in interval i_s is then determined as the largest number of shifts j that can be active that yields a total staffing cost of at most c_w^{init}:

$$s_{i_s}^{\mathsf{UB}} = \left\lfloor \frac{c_w^{\mathsf{init}}}{c_a^{j_{\mathsf{min}}}} \right\rfloor, \forall i_s \in \mathbf{I}_s \, .$$

All solutions for which $s(i_s) > s^{\mathsf{UB}}(i_s)$ in at least one staffing interval yield a staffing cost that exceeds c_w^{init}, and should not be considered in the search tree.

Appendix C: Simulated performance metrics

The probability of excessive waiting, $\Pr(W_t > \tau)$, is measured from the simulation model by means of *virtual waiting times*. The virtual waiting time corresponds to the time between t and the earliest time at which a (scheduled) server becomes available, because all customers that arrived before t have been served (Gross et al., 2008; Le Minh, 1978; Mandelbaum and Momğilović, 2008):

$$W_t = \min\{w : \left(N_{t+w}^t \leq s_{t+w} - 1\right) \wedge (w \geq 0)\},$$

where N_{t+w}^t denotes the number of customers that arrived before time t that are still in system at time $t + w$. Note that the virtual waiting time is measured at a particular time instant (as opposed to *observed* waits, which are measured over an interval). The virtual waiting time distribution can be measured in a straightforward way through simulation. We insert a virtual (dummy) customer into the system at each time $t \in \mathbf{t}_p$ in replication r, such that the virtual waiting time W_t^r equals the time at which this dummy customer would enter service. Let R represent the total number of replications in the simulation run. Define δ_t^r as a binary variable that signals whether the virtual waiting time exceeds the target τ for a given time t and replication r:

$$\delta_t^r = \begin{cases} 1 & \text{if } W_t^r > \tau, \\ 0 & \text{otherwise} \end{cases} .$$

The probability of excessive waiting at time t then can be evaluated as:

$$\Pr(W_t > \tau) = \frac{1}{R} \sum_{r=1}^{R} \delta_t^r \, .$$

Appendix D: Shift specifications

Staffing interval length (number of shifts)	Shift specification {start time, end time, start time break}
$\Delta_s = 240$ ($W = 5$)	$\{0,4,-\}$, $\{4,8,-\}$, $\{8,12,-\}$, $\{0,8,-\}$, $\{4,12,-\}$
$\Delta_s = 120$ ($W = 12$)	$\{0,4,-\}$, $\{2,6,-\}$, $\{4,8,-\}$, $\{6,10,-\}$, $\{8,12,-\}$, $\{0,6,-\}$, $\{2,8,-\}$, $\{4,10,-\}$, $\{6,12,-\}$, $\{0,8,-\}$, $\{2,10,-\}$, $\{4,12,-\}$
$\Delta_s = 60$ ($W = 45$)	$\{0,4,-\}$, $\{1,5,-\}$, $\{2,6,-\}$, $\{3,7,-\}$, $\{4,8,-\}$, $\{5,9,-\}$, $\{6,10,-\}$, $\{7,11,-\}$, $\{8,12,-\}$, $\{0,6,2\}$, $\{1,7,3\}$, $\{2,8,4\}$, $\{3,9,5\}$, $\{4,10,6\}$, $\{5,11,7\}$, $\{6,12,8\}$, $\{0,6,3\}$, $\{1,7,4\}$, $\{2,8,5\}$, $\{3,9,6\}$, $\{4,10,7\}$, $\{5,11,8\}$, $\{6,12,9\}$, $\{0,6,4\}$, $\{1,7,5\}$, $\{2,8,6\}$, $\{3,9,7\}$, $\{4,10,8\}$, $\{5,11,9\}$, $\{6,12,10\}$, $\{0,8,3\}$, $\{1,9,4\}$, $\{2,10,5\}$, $\{3,11,6\}$, $\{4,12,7\}$, $\{0,8,4\}$, $\{1,9,5\}$, $\{2,10,6\}$, $\{3,11,7\}$, $\{4,12,8\}$, $\{0,8,5\}$, $\{1,9,6\}$, $\{2,10,7\}$, $\{3,11,8\}$, $\{4,12,9\}$

Table 6: Shift specifications (all breaks are assumed to be 1 hour). W represents the size of the shift set for problem instances with staffing interval length Δ_s

References

Agresti, A., B. A. Coull. 1998. Approximate Is Better than "Exact" for Interval Estimation of Binomial Proportions. The American Statistician 52(2) 119-126.

Atlason, J., M.A. Epelman, S.G. Henderson. 2004. Call Center Staffing with Simulation and Cutting Plane Methods. Annals of Operations Research 127(1) 333-358.

Atlason, J., M.A. Epelman, S.G. Henderson. 2008. Optimizing Call Center Staffing Using Simulation and Analytic Center Cutting-Plane Methods. Management Science 54(2) 295-309.

Campello, F., A. Ingolfsson. 2011. Exact Necessary Staffing Requirements based on Stochastic Comparisons with Infinite-Server Models. Working paper, University of Alberta.

Castillo, I., T. Joro, Y.Y. Li. 2009. Workforce scheduling with multiple objectives. European Journal of Operational Research 196(1) 162-170.

Creemers, S., M. Defraeye, I. Van Nieuwenhuyse. 2013. A Markov model for measuring service levels in nonstationary $G(t)/G(t)/s(t) + G(t)$ queues. Research report KBI_1306, KU Leuven, Leuven, Belgium.

Dantzig, G. 1954. A comment on Edie's traffic delay at toll booths. Operations Research 2 339-341.

Defraeye, M., I. Van Nieuwenhuyse. 2011. Setting staffing levels in an emergency department: opportunities and limitations of stationary queuing models. Review of Business and Economics 56(1) 73-100.

Defraeye, M., I. Van Nieuwenhuyse. 2013. Controlling excessive waiting times in small service systems with time-varying demand: An extension of the ISA algorithm. Decision Support Systems 54(4) 1558-1567.

Ernst, A.T., H. Jiang, M. Krishnamoorthy, D. Sier. 2004. Staff scheduling and rostering: A review of applications, methods and models. European Journal of Operational Research 153(1) 3-27.

Feldman, Z., A. Mandelbaum, W.A. Massey, W. Whitt. 2008. Staffing of Time-Varying Queues to Achieve Time-Stable Performance. Management Science 54(2) 324-338.

Grassmann, W.K., Transient solutions in markovian queueing systems, Computers & Operations Research 4(1) (1977) 47-53.

Green, L.V., J. Soares. 2007. Computing time-dependent waiting time probabilities in $M(t)/M/s(t)$ queueing systems. Manufacturing & service operations management 9(1) 54-61.

Gross, D., J.F. Shortle, J.M. Thompson, C.M. Harris. 2008. Fundamentals of Queueing Theory (4th Edition). Wiley Series in Probability and Statistics, Wiley-Blackwell.

Guan, Y. 2012. A generalized score confidence interval for a binomial proportion. Journal of Statistical Planning and Inference 142 785–793.

Helber, S., K. Henken. 2010. Profit-oriented shift scheduling of inbound contact centers with skills-based routing, impatient customers, and retrials. OR Spectrum 32(1/4) 109-134.

Henderson, S.G., A.J. Mason. 1998. Rostering by iterating integer programming and simulation. Winter Simulation Conference Proceedings (WSC'98) 1 677-683.

Henderson, S.G., A.J. Mason, I. Ziedins, R. Thomson. 1999. A Heuristic for Determining Efficient Staffing Requirements for Call Centres. Working paper, University of Auckland, New Zealand.

Ingolfsson, A., A. Haque, A. Umnikov. 2002. Accounting for time-varying queueing effects in workforce scheduling. European Journal of Operational Research 139(3) 585-597.

Ingolfsson,A., E. Akhmetshina, S. Budge, Y. Li. 2007. A survey and experimental comparison of service level approximation methods for non-stationary $M(t)/M/s(t)$ queueing systems with exhaustive discipline, INFORMS Journal on Computing 19(2) 201-214.

Ingolfsson, A., F. Campello, X. Wu, E. Cabral. 2010. Combining integer programming and the randomization method to schedule employees. European Journal of Operational Research 202(1) 153-163.

Ingolfsson, A. 2005 Modeling the $M(t)/M/s(t)$ queue with an exhaustive discipline. Working paper, University of Alberta, Canada. Available online on http://www.business.ualberta.ca/aingolfsson/publications.htm

Izady, N., D.J. Worthington. 2012. Setting staffing requirements for time-dependent queueing networks: The case of accident and emergency departments. European Journal of Operational Research 219 531-540.

Koole, G., E. van der Sluis. 2003. Optimal Shift Scheduling with a Global Service Level Constraint. IIE Transactions 35(11) 1049-1055.

Law, A.M., W.D. Kelton. 2000. Simulation modeling and analysis. McGraw-Hill series in industrial engineering and management science, McGraw-Hill (Boston).

Le Minh, D. 1978. The discrete-time single-server queue with time-inhomogeneous compound Poisson input and general service time distribution. Journal of Applied Probability, 590-601.

Mandelbaum, A., P. Momğilović. 2008. Queues with many servers: The virtual waiting-time process in the QED regime. Mathematics of Operations Research 33(3) 561-586.

Mandelbaum, A., W.A. Massey. M.I. Reiman, A. Stolyar, B. Rider. 2002. Queue lengths and waiting times for multiserver queues with abandonment and retrials. Telecommunication Systems 21(2-4) 149-171.

Newcombe, R. G. 1998. Two-sided confidence intervals for the single proportion: comparison of seven methods. Statistics in Medicine 17 857–872.

Sinreich, D., O. Jabali. 2007. Staggered work shifts: a way to downsize and restructure an emergency department workforce yet maintain current operational performance. Health Care Managent Science 10 293-308.

Thompson, G.M. 1993. Accounting for the multi-period impact of service when determining employee requirements for labor scheduling. Journal of Operations Management 11(3) 269-287.

Thompson, G.M. 1997. Labor Staffing and Scheduling Models for Controlling Service Levels. Naval Research Logistics 44(8) 719-740.

Whitt, W. 2007. What you should know about queueing models to set staffing requirements in service systems. Naval Research Logistics 54(5) 476-484.

FACULTY OF ECONOMICS AND BUSINESS
Naamsestraat 69 bus 3500
3000 LEUVEN, BELGIË
tel. + 32 16 32 66 12
fax + 32 16 32 67 91
info@econ.kuleuven.be
www.econ.kuleuven.be